An Improved Apriori Algorithm on the Frequent Itemse

Xiang Fang

Department of Department of Computer Science, Shandong Institute of Business and Technology, Shandong, 264005, China

email:fangxiang782@sina.com

Keywords: Association Rule; Apriori Algorithm; Frequent Itemse

Abstract. According to the problem that the traditional Apriori algorithm needs to scan database frequently, an improved strategy and corresponding algorithm is put forward in this paper. This method, only when L_1-candidates are produced, scans the transaction database D. The rest frequent itemsets are produced through the scan of preceding result in place of the transaction database. The improved algorithm reduces I/O load and has higher efficiency.

Introduction

A subfield of computer science, Data mining (the analysis step of the "Knowledge Discovery in Databases" process, or KDD [1]), is the computational process of extracting information and transforming it into an understandable patterns from a large data sets involving combining methods from artificial intelligence, machine learning, statistics, database systems and so on [2]. And the association rule, a well researched method for discovering interesting relations between the items in large databases, is popular and effective for decision-making in financial mathematics, medical, intrusion detection and Web analysis in recent years [3][4]. Apriori algorithm is the most classic algorithm for association rule on transactional databases. It proceeds by identifying the frequent items in the database and extending them to an itemset as long as those itemsets appear sufficiently often in the database.

Since the Apriori algorithm had been brought by R.Agrawal in 1993 [5], scholars have put forward lots of algorithmic improvements, such as AprioriTid, AprioriHybrid [6], DIC [7], DHP [8]. The improved algorithms can be divided into two kinds based on whether produces the candidate itemset: One is quite direct-viewing and easy to realize, but they have to produce the candidate itemset and scan the transactional database repetitiously, the classic Apriori algorithm is the most famous one of them [9]; the other is very efficient, as it does not need to produce candidate itemset, but they is very complex, because the association rules are found by the pattern tree. The Han's fp-growth algorithm is the most influential one [10].

Taking into account deficient of the first kind, many scholars proposed algorithmic development, such as a row optimization based on disperses, business-compresses, business-division, optimization based on the sampling, optimization based on the dynamic itemset and so on[11]. Unfortunately, those improved algorithms have to produce the massive candidate itemsets. This paper unifies the two thoughts and puts forward the improved strategy and corresponding algorithm to the traditional Apriori algorithm: Find the set L_k(frequent itemsets) from L_{k-1} not C_{k+1} (candidate itemsets) and simultaneously remove the candidate items which have infrequent patterns.

Traditional Apriori Algorithm and Defect of It

Let $I=\{i_1, i_2, \dots , i_m\}$ denote the set of items that are displayed in a store. Moreover, let D represent a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A unique identifier, namely TID, is associated with each transaction. Let A is a set of item in I, A transaction T is said to contains A if $A \subseteq T$. An association rules is an implication of the form $A \rightarrow B$, where $A \subset I$, $B \subset I$ and $A \cap B \neq \Phi$. Support of rule, denoted by s, is the percentage of transactions in D that contain A also contain B, it can be computed by $|AB|/|D|$; Confidence of rule, denoted by c, is the ratio between the number of transactions containing both A and B and the number of transactions containing A, it can be computed as |AB|/|A|. In database D, a set of items is called itemset, and an itemset that contains k items is called k-itemset. Support count is number of transactions containing an itemset, if an itemset whose support is greater than or equal to a minimum support threshold (called minsup), we name it the frequent itemset.

Apriori algorithm finds frequent itemsets, and generates successively longer candidate itemsets from shorter ones that are known. Each produce of candidate itemset requires scanning the dataset to determine whether its frequency exceeds the minsup. Apriori algorithm can find rules in the transaction database with the known minsup and minimum confidence through the following two sub-questions[12]:

(1)Find the frequent itemset L_k from the candidate itemset C_{k+1}, and scan the database to remove infrequent patterns according to a predetermined minimum support count.

(2)Generate the interesting association rule from the frequent itemset.

The overall performance of association rules is determined by the first question, the following is the description of the algorithm:

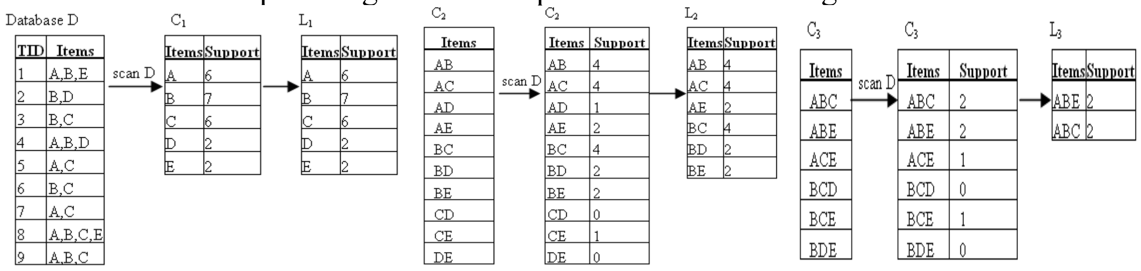
```
Input: transaction database D, minsup
Output: the set of Frequent L in the database D
Method:
 (1)  $L_1$={large1-itemsets};
 (2)  for (k=2;$L_{k-1}\neq\phi$;k++) do
 (3)       {$C_k$=Apriori_gen( $L_{k-1}$,min-sup) ;            // Produce $L_k$ by $L_{k-1}$
 (4)        for each transaction $t\in D$                      // Compute the support
 (5)             {$C_t$=subse($C_k$,t) ;                       // Find candidate
 (6)               for each candidate $c\in C_t$ do
 (7)               c.count++;
 (8)             }
 (9)        $L_k$={$c\in C_k$|c.count≥min-sup}
(10)       }
(11)  return L=$\cup_k L_k$;
```

There is a transaction database D, which has nine transactions, as shown in Table 1.

Tab1.Transaction database D

TID	Itemset
1	A,B,E
2	B,D
3	B,C
4	A,B,D
5	A,C
6	B,C
7	A,C
8	A,B,C,E
9	A,B,C

By setting the minimum support as 20%, namely the support is 2. Find frequent itemset for D by means of the traditional Apriori algorithm. The process is shown in Figure1.



TID	Items
1	A,B,E
2	B,D
3	B,C
4	A,B,D
5	A,C
6	B,C
7	A,C
8	A,B,C,E
9	A,B,C

Items	Support
A	6
B	7
C	6
D	2
E	2

Items	Support
A	6
B	7
C	6
D	2
E	2

Items
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE

Items	Support
AB	4
AC	4
AD	1
AE	2
BC	4
BD	2
BE	2
CD	0
CE	1
DE	0

Items	Support
AB	4
AC	4
AE	2
BC	4
BD	2
BE	2

Items
ABC
ABE
ACE
BCD
BCE
BDE

Items	Support
ABC	2
ABE	2
ACE	1
BCD	0
BCE	1
BDE	0

Items	Support
ABE	2
ABC	2

Fig1. The operation process of the traditional Apriori algorithm

By the analysis, we can see, traditional Apriori algorithm has two deadly bottlenecks [4]:

(1) Scans the very big database repeatedly to produce L_K increase I/O load and reduce efficiency. Each item in the candidate itemsets must scan database one time to decide whether it can be joined to the L_k. So it needs to scan the transaction database as the same number as the elements of the frequent itemset.

(2) When it carries on the k-th scanning, the algorithm does not use the former result.

At the same time, the improved algorithms mentioned above are improvement or extension based on architecture of the traditional algorithm, the efficiency has not been much improved. The key of the improved algorithm that we propose in his paper is how to reduce the scan through the results of previous scan.

Improved Apriori Algorithm

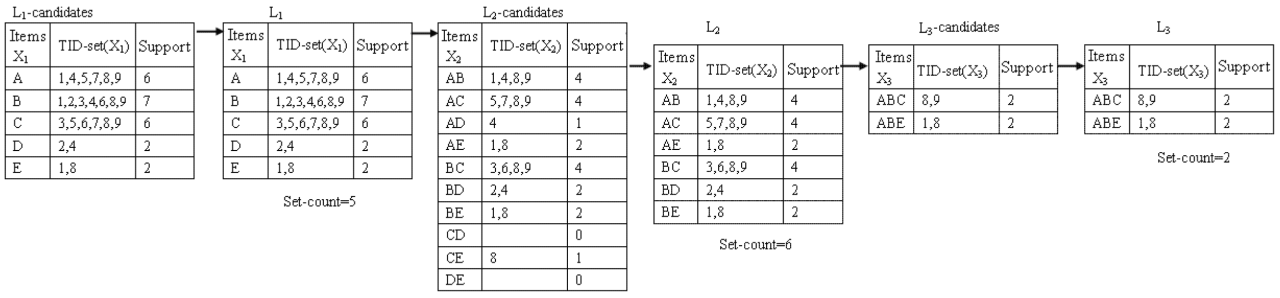
Let TID-set(A) denote the set of transaction TID which contain item A in D, so the amount of transaction which contain A in D is the amount of item of TID-set(A), more exactly, sup-count(A) can be computed by |TID-set(A)|. The transaction set in D, which have A and B, is the intersection of TID-set(A) and TID-set(B), and sup-count($A \rightarrow B$) can be computed as |TID-set(A)∩TID-set(B)|. The join and the pruning of Apriori algorithm is improved correspondingly: all non-empty subset of frequent itemset must be frequent; to produces L_k-candidates, we only join the set whose fore k-2 item is same in L_{k-1}. So we propose the following improvement to the traditional Apriori algorithm:
(1) Compute minimum sup-count by min-sup*|D|.
(2) Scan the database once to produce L_1-candidates, simultaneously construct TID-set(X_1) for each item. After scanning, compute sup-count for each item and find the set of frequent items L_1.
(3) Produces L_2-candidates from L_1*L_1.Scanning L_1, we can find TID-set(X_2) and sup-count of each itemset, deletes the patterns whose frequencies don't satisfy the min support count, and find L_2.
(4) In order to produce $L_k(k\geq 3)$, join itemsets which satisfy the join rule. Scanning $L_{k-1,}$ we can find TID-set(X_k) and compute sup-count of each itemset, then deletes the patterns whose frequencies do not satisfy the minimum support count, and find L_k.

The pseudo code of Algorithm is as follows:

```
Input: transaction database D; min-sup.
Output: the set of Frequent L in the database D
Method:
(1)min-sup-count=min-sup*|D|
(2)$L_1$-candidates=find all one itemsets(D)          //Scan D and produce $L_1$-candidates
(3)$L_1$={<$X_1$,TID-set($X_1$)>∈$L_1$-candidates |sup-count≥min-sup-count}
(4)for (k=2; $L_{k-1}\neq\phi$;k++) do {
(5)       {for each k-itemset ($x_i$,TID-set($x_i$)∈$L_{k-1}$ do
(6)              for each k-itemset ($x_j$,TID-set($x_j$)∈$L_{k-1}$ do
(7)                  if ($x_i[1]=x_j[1]$)∧($x_i[2]=x_j[2]$)∧…∧($x_i[k-2]=x_j[k-2]$ then
(8)                       {$L_k$-candidates.$X_k$= $X_i$* $X_j$;
(9)                            $L_k$-candidates.TID-set($X_k$) = TID-set($X_i$)∩TID-set($X_j$)
(10)          }}
(11)          for each k-itemset<$X_k$,TID($X_k$)>∈$L_k$-candidates do
(12)                   sup-count=|TID-set|
(13)          $L_k$={<$X_k$,TID-set($X_k$)>∈$L_k$-candidates|sup-count≥min-sup-count}
(14) }
(15) set-count=$L_k$.itemcount                        //updata set-count
(16) return L=$\cup_k L_k$;
```

We generate L_k through the Line 5 to 14, thereinto line 5 to 10 generate L_k-candidate, line 7 judge whether the fore k-2 item is same of L_{k-1}, if the same, we join the two L_{k-1}, and add it to L_k-candidate. Line 11 to 13 compute the support for each items and deletes the patterns whose frequencies do not satisfy the minimum support count. Line 15-16 compute item-count and input the result.

In order to specify the discussion above, we give an application. According to the algorithm, first we produce L_1-candidates and find the set of frequent items L_1. Sele-join L_1 to produce L_2-candidates, scan L_1, and achieve TID-set(X_1) and sup-count, deletes the patterns whose frequencies do not satisfy the minimum support count,. and find L_2. By joining the itemset which satisfied rule in L_2, we can obtain {{A,B,C}, {A,B,E}, {B,C,D}, {B,C,E}, {A,C,E}, {B,D,E}}. However {A,D}, {C,D}, {C,E}, and {D,E} is not the frequent, and {B,C,D}, {B,C,E}, {A,C,E}, {B,D,E} don't belong to L_3-candidates. Compute TID-set(X_3) and the support count separately, L_3 is composed by itemsets which have the minimum support count. The itemset which satisfies the condition in L_3 sele-join, and obtains {{A,B,C,E}}. {B,C,E} is not the frequent itemset, so L_4-candidates=ϕ, the algorithm finishes. The process is as shown in Figure2.



L_1-candidates

Items X_1	TID-set(X_1)	Support
A	1,4,5,7,8,9	6
B	1,2,3,4,6,8,9	7
C	3,5,6,7,8,9	6
D	2,4	2
E	1,8	2

L_1

Items X_1	TID-set(X_1)	Support
A	1,4,5,7,8,9	6
B	1,2,3,4,6,8,9	7
C	3,5,6,7,8,9	6
D	2,4	2
E	1,8	2

Set-count=5

L_2-candidates

Items X_2	TID-set(X_2)	Support
AB	1,4,8,9	4
AC	5,7,8,9	4
AD	4	1
AE	1,8	2
BC	3,6,8,9	4
BD	2,4	2
BE	1,8	2
CD		0
CE	8	1
DE		0

L_2

Items X_2	TID-set(X_2)	Support
AB	1,4,8,9	4
AC	5,7,8,9	4
AE	1,8	2
BC	3,6,8,9	4
BD	2,4	2
BE	1,8	2

Set-count=6

L_3-candidates

Items X_3	TID-set(X_3)	Support
ABC	8,9	2
ABE	1,8	2

L_3

Items X_3	TID-set(X_3)	Support
ABC	8,9	2
ABE	1,8	2

Set-count=2

Fig2. The operation process of the improved Apriori algorithm

Conclusion

The biggest drawback of traditional Apriori algorithm is that each item in the candidates of frequent item sets must scan database one time, and the efficiency of execution is very low. we proposed an improvement, this algorithm scan the transaction database D only when L_1-candidates is produced, the rest scan the preceding result instead of the transaction database, greatly reducing I/O load and increase efficiency. The interesting association rule can be accessed more effectively by the improved Apriori algorithm.

References

[1] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery in Database [J]. American Association for Artificial Intelligence, AAAI Press,July,1996, 37-54.

[2] M.S.Chen, J.Han, and P.S. Yu. Data Mining: An Overview from Database Perspective [J]. Knowledge and Data Engineering, 1996 (8) 866-883.

[3] A.Chen, N.Chen, L.X.Zhou. The Technology and Appliance of Data Mining[M]. BeiJing: Science Press 2006.

[4] W.W. Chen. Data Warehouse and Data Mining[M]. BeiJing: Tsinghua University Press 2006.

[5] J. Cheng, Y.P. Ke, Wilfred Ng. Effective Elimination of Redundant Association Rules [J]. Data Mining and Knowledge Discovery. 2008 (16) 221-249

[6] R.Agrawal, and R.Srikant. Fast Apriorithms for Mining Assiciation Rules [A]. Proc. the 20th VLDB[C]. 1994 487-499.

[7] Brin S, Motwai R, Ullman J D, Tsur S. Dynamic Itemset Counting and Implication Rules for market basket data[A]. Proceedings of the ACM SIGMOD International Conference on Management of Data[C].1997 255–264

[8] J.S.Park, M.S. Chen, and P.S. Yu. An Effective Hash Based Algorithm for Mining Association Rules[A]. Proc.ACM SGMOD[C]. 1995 175-185.

[9] H.Shen, B.L.Lu, M.S. Utiyama. Comparison and Improvements of Feature Extraction Methods for Text Categorization [J]. Computer Simulation, 2006 (23) 222-224.

[10] L.L Dai, H.Y. Huang, and Z.X. Chen. A Comparative Study on Feature Selection in Chinese Text Categorization [J]. Journal of Chinese Information Processing , 2004(18) 26-32.

[11] Q. Zhou, M.S. ZHAO. Study on Feature Selection in Chinese Text Categorization [J]. Journal of Chinese Information Processing, 2004 (18) 17-23.

[12] H.X Chai, Y.Wang. Improvement of Apriori Algorithm[J]. Computer Engineering and Applications, 2007 (43) 158-161.